

24. Implementing the FFT and Multiplying Numbers

1. Introduction

We will first describe in detail how a spreadsheet can be set up to perform the Fast Fourier Transform algorithm.

We will then apply it to the task of multiplying large numbers.

2. Implementing FFT's on a Spreadsheet

We will describe the fast algorithm for evaluating a polynomial of degree up to $n-1$ at n roots of unity.

This algorithm can be applied in any field in which the equation x^n-1 has a primitive root.

Such fields include the complex numbers, and numbers mod any prime p of the form $qn+1$.

The task of programming this algorithm is roughly the same for n any power of 2. We will create a spreadsheet for performing the FFT with $n=16$.

We can do so mod 17 or 97 or 257, which are all fields that possess 16^{th} roots of unity. We will arrange our algorithm so that we can switch moduli and primitive roots easily which will allow us to perform our operations mod each of these. The Chinese remainder theorem will then allow us to recover evaluations as high as the product of all three of these moduli.

The goal of this algorithm is to obtain evaluations of our polynomial at arguments $z^0 (-1), z^1, z^2, \dots, z^{n-1}$, where z is a primitive n -th root of unity on our field. This can be done in the complex plane where we can use $z=\cos(2\pi/n)+isin(2\pi/n)$.

We will instead illustrate the procedure mod 17, using 3 as our primitive 16^{th} root of unity.

Since we want to be able to copy our work and easily substitute different moduli and roots, we prepare our spreadsheet by putting the modulus, 17, in square in the first row and column of the space we devote to it, and put $\text{=(square to the left)}$ in the 15 squares to its right.

Similarly, we put 3 in the row beneath it. In this way if we copy our work to different columns in the same rows, we will be able to insert a different modulus and root in each copy by changing two entries.

In our third row we enter indices from 0 to 15 which will represent the power of x whose coefficient in our polynomial p is to be entered into the fourth row in that column. When we reach the end of the algorithm, the evaluation at z^k will appear in the column with index k .

So when we begin, our spreadsheet entries will look like

17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	3	5	2	5	6	2	1	3	2	1	4	6	4	3	1

where the last row represents the polynomial that starts as $1 + 3x + 5x^2 + 2x^3 + \dots$

Notice that, while this is not a particularly unnatural way to describe a polynomial, it is not the normal way, which starts with the highest power first. Here the highest power appears on the right.

The algorithm will consist of 4 steps, which corresponds to the fact that 16 is 2^4 ; if we perform a 2^k FFT we will need k steps.

For convenience we label the columns A_0 to A_{15} here. On a spreadsheet these would be A through Q or any similar range.

STEP 1

We begin by entering in $A_05: = \text{mod}(A_04+A_84,A_0\$1)$
 We copy this into A_{15} through A_75 . These represent the even power evaluations. We must then move them to the even power locations; the entry in A_{j5} should therefore be moved to square A_{2j5} for each j .

We then enter in $A_06: = \text{mod}((A_04-A_84)*A_0\$2^{A_0\$3},A_0\$1)$ and copy it into A_{16} through A_76 . These entries must then be moved to the right of the entries in the even places in row 5.

STEP 2

You may copy the entry in A_05 down into A_06 and then across into A_{16} - A_76 . These should then be moved so as to occupy the squares in the 6th row with indices 0-1, 4-5, 8-9, and 12-13.

You may then enter into $A_07: \text{mod}((A_05-A_85)*A_0\$2^{(2*\text{int}(A_0\$3/2))},A_0\$1)$, copy these into the 7 places to its right, and move these into the 6th row in columns with indices 2-3,6-7,10-11, and 14-15.

STEP 3

Copy the entry in A_06 down into A_07 and then across into A_{17} - A_77 . These should then be moved so as to occupy the squares in the 7th row with indices 0-3 and 8-11.

Enter into A₀₈: mod ((A₀₆-A₈₆)*A₀\$2^(4*int(A₀\$3/4)),A₀\$1) , copy this into the 7 places to its right, and move these into the 7th row in columns with indices 4-7, and 12-15.

STEP 4

Copy the entry in A₀₇ down into A₀₈ and then across into A₁₈- A₇₈+

Enter into A₀₉: mod (A₀₆-A₈₆,A₀\$1) , copy this into the 7 places to its right, and move these into the 8th row in columns with indices 8-15.

At this point the entries in the 8th row here should be the evaluations sought.

Please realize that there are a large number of ways that careless errors can creep into this procedure, and it is absolutely essential that you check that it works before considering yourself finished with it.

To check it you can copy rows 4 to 8 anywhere directly below A₀-A₁₅, put =A₀₈ in the entry just above the copy of row 4 in column 0, and copy that entry across that row.

If your spreadsheet is correct the sum of the entry in row 8 in the column with index j should be the negative of the entry in your bottom row in column with index 16-j for j>0. (And the two entries in the 0 column should add up to 0 mod 17.

If you find an error, you should try to locate it, by putting in a simple input polynomial, such as 1 or x or x², etc., and checking whether your 8-th line is what it should be for that polynomial. If it is not you can trace back to find the entry or entries that are wrong.

With a little practice you will be able to locate each bug quickly. You will feel good when you are done.

(If you really wanted to save machine effort, you would precalculate the powers of z that each odd power is to be multiplied by, and have them listed appropriately in rows to make the multiplication convenient. As it is, my machine calculates mod(z⁷,p) by first raising z to the 7th, and then dividing by p and taking the remainder. With larger n values this sort of thing could lead to a huge number, removing any benefit from use of the method.)

FFT 16

17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	3	5	2	0	6	2	1	0	2	1	4	6	4	0	1
1	1	5	3	6	2	6	14	6	7	10	10	2	13	2	0
7	8	12	11	15	13	6	5	8	15	1	10	8	14	9	6
15	6	13	4	16	10	11	1	6	10	15	11	6	4	12	4
4	16	11	15	5	14	6	5	9	13	15	10	10	6	16	14

FFT	16															
	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	
	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	4	16	11	15	5	14	6	5	9	13	15	10	10	6	16	14
	13	12	12	9	9	15	8	16	15	3	3	6	5	3	2	3
	11	15	15	9	15	15	13	10	14	1	1	3	10	2	5	8
	8	16	16	12	14	14	14	6	8	0	1	1	14	16	2	9
	16	16	0	13	11	13	16	15	0	16	15	11	0	15	12	14

3. Application to Multiplying Numbers

The naïve and standard approach to multiplying numbers is to multiply each digit of each number by one another, and put the results together in a standard way. This involves n^2 operations, and we can try to find a method that for extremely large numbers will be better. The FFT provides a way to do this.

A number, written in standard form is actually being represented as a polynomial, evaluated at the base of the numbering system. When we were dealing with polynomial remainders, we used as identifier of a remainder the number which that remainder gave when evaluated at argument 2. A number like 892 actually means $8 \cdot 10^2 + 9 \cdot 10^1 + 2 \cdot 10^0$.

Thus we can multiply two numbers, by choosing some base, representing them as polynomials in this way, and multiplying the polynomials.

This step is where the bulk of the work is needed and here is where we can use the FFT. What we will first do is evaluate the polynomials at roots of unity. Then we will take the products of the polynomials there, which will give the values of the product polynomial at the roots of unity. We will then take the inverse FFT of this to get the polynomial coefficients themselves.

The resulting coefficients will not usually be between 0 and $b-1$ where b is the base, since they are sums of products of coefficients of the factor polynomials and could be much greater than $b-1$. We will have to perform a carrying operation to retrieve them.

If we do our evaluations mod 17 and mod 97, for example, we can handle coefficients in our product polynomial that are as large as $17 \cdot 97 - 1$, and once we are set up to do FFT's mod 17 we can copy so as to do them mod 97 as soon as we find a primitive 16^{th} root of unity mod 97.

Please notice that the nFFT can handle a product polynomial of degree up to $n-1$. That means that the sum of the degrees of the factor polynomials can be at most $n-1$. Thus a 16FFT can handle the product of a polynomial of degree 8 multiplied with one of degree 7.

To find such a product, with coefficients to base 10, you can **set up three FFT spreadsheets in each of two sets of columns, one set for modulus 17, the other for 97.**

You enter the polynomials representing **your factors** (in the increasing power order) in the **first two** of these, and **the product of the outputs to each** (mod the appropriate modulus) **as input to the third.**

Next you multiply each entry in the output row by the inverse of 16 mod 17 and mod 97 to get the coefficients of the product polynomial mod 17 and 97

Then **if you move the output of the third spreadsheet in the 0 column into a 16th column** for each you will have the coefficients arranged **in the standard decreasing power order for polynomials mod 17 and mod 97.**

There are two remaining steps.

First you must **find the coefficients of the product polynomial mod 17*97 from their values mod 17 and 97.**

Second you must perform **the carrying** necessary to put the coefficients in standard form (that is, from 0 to 9)

Neither of these involve much effort, but they must be done, and here is how:

4. Finding $A \bmod B \cdot C$ given $A \bmod B$ and $A \bmod C$ with B and C relatively prime

Suppose we have discovered that a coefficient x obeys

$$x = a \bmod 17 \text{ and } x = b \bmod 97$$

We want to know the actual value of x . We know that x is the product of at most 8 terms each of which is at most $9 \cdot 9$ or 81 so that x lies somewhere between 0 and 648. This means we determine x completely if we know it mod $17 \cdot 97$.

The first equation here tells us for some unknown A ,

$$x = a + 17A,$$

and the second yields

$$x = b + 97B.$$

We can subtract these equations, to find

$$(a-b) = 97B - 17A.$$

Now we can work Euclid's algorithm backwards here to find

$$1 = 97C - 17D$$

from which we can deduce $(a-b) = 97C(a-b) - 17D(a-b)$, or $A=D(a-b)$ and we have

$$x = (a + 17*D*(a-b)) \bmod 97*17.$$

Here $-D$ is the inverse of $17 \bmod 97$.

Similar relations apply for any two prime moduli.

5. Carrying

At this point we know our coefficients but they must be converted to numbers from 0 to 9. Let us call the coefficient in the $(16-j)$ -th column $c(j)$.

We start with $c(0)$ (in column 16) and we set $d(0) = c(0)$. Let us call the actual least significant digit of our product $n(1)$ and the next $n(2)$ and so on,

We then have $n(j) = \text{mod}(d(j-1), 10)$ and $d(j) = c(j) + \text{int}(d(j-1)/10)$.

Here $n(j)$ gives the actual j -th digit of the product and carrying is the act of computing $d(j)$ from $c(j)$.

For example, suppose we get coefficients 9 16 23 11 in decreasing order of significance. Then we find $n(1) = 1$, $d(1) = 24$ so that $n(2)=4$, $d(2)=18$, $n(3) = 8$, $d(3) = 10$, $n(4)= 0$, $d(5)=1$, $n(5)=1$, and the number obtained is 10841.

5. Summary

To apply this algorithm here you need to perform the following steps.

1. Find a (small) primitive 16^{th} root of unity mod 97.
2. Find the inverse of 16 and of 17 mod 97.
3. Set up your 16FFT spreadsheet and copy it 6 times appropriately.
4. Enter your numbers into appropriate inputs copies
5. Enter the products into the appropriate inputs
6. Move column 0 to column 16 in both product outputs
7. Find the coefficients mod $97*17$ using the inverse of $17 \bmod 97$
8. Perform the carrying necessary to extract an ordinary number from your answer.

Exercise: Do all these steps and succeed in multiplying 12345678 by 987654321 this way.

Let us call the co

We can accomplish this by starting with the least significant coefficient, here the 16^{th} which we will call $c(16)$